Wiki Content Templating

Angelo Di Iorio
Computer Science Department
University of Bologna
Mura Anteo Zamboni, 7
40127 Bologna, ITALY
diiorio@cs.unibo.it

Fabio Vitali
Computer Science Department
University of Bologna
Mura Anteo Zamboni, 7
40127 Bologna, ITALY
fabio@cs.unibo.it

Stefano Zacchiroli
Computer Science Department
University of Bologna
Mura Anteo Zamboni, 7
40127 Bologna, ITALY
zacchiro@cs.unibo.it

ABSTRACT

Wiki content templating enables reuse of content structures among wiki pages. In this paper we present a thorough study of this widespread feature, showing how its two state of the art models (functional and creational templating) are sub-optimal. We then propose a third, better, model called lightly constrained (LC) templating and show its implementation in the Moin wiki engine. We also show how LC templating implementations are the appropriate technologies to push forward semantically rich web pages on the lines of (lowercase) semantic web and microformats.

Categories and Subject Descriptors

I.7.2 [**Document Preparation**]: Hypertext/hypermedia; H.3.5 [**Online Information Services**]: Web-based services

General Terms

Design

Keywords

wiki engine, template, semantic web, microformats

1. INTRODUCTION

The Web is the largest and more lively source of information encoded in a machine readable format. It is also mostly useless without a downstream human understanding of a rendered version of the contained pages. The Web's natural evolution—the *Semantic Web* [1] (with capital "S" and "W")—aims to change the encoded information so that it is not only machine readable, but also machine processable, permitting the creation of added-value services based on the semantics of web pages.

The high footprint of most technologies related to the Semantic Web has so far hindered its diffusion; to counter this, two recent trends are trying to lower the technological barriers that have so far kept authors away from semantic annotating their content: microformats and semantic wiki. On one hand *microformats* [7, 8] and similar enabling technologies (e.g. RDFa [14]) permit one to express tangible metadata as part of the content markup of web pages. Taken together those technologies are often referred to as *semantic web* (with lowercase "s" and "w"). On the other hand, *semantic wikis* (e.g. [11, 16, 17]) exploit the content mediation performed by the wiki engine to deliver semantically rich web pages and give to authors simple interfaces (sometimes even invisible interfaces based on syntactic quirks!) to semantically annotate their content or to import semantic metadata.

Though (lowercase) semantic web and semantic wikis fill a technological gap inhibiting authors to provide metadata, the question of how to actually *foster* the creation of semantically rich web pages is still open. By analogy, we observe that a similar goal has been traditionally pursued by wiki content templating mechanisms[1] which have been available since the first wiki clones under the name of seeding pages. The key idea of this work is indeed expressed by the intuition *wiki template = information pattern*: (content) templates are used in wikis to give authors skeletons (or patterns) to be filled with precise information. Common use cases of templates are coherent with this intuition:

1. information and navigation boxes in Wikipedia[2] are used to collect sets of information about software (name, homepage URL, license, version, ...) or musician albums (title, genre, year, previous album, ...) which will then be rendered as part of the page;

2. seeding pages are used to foster the creation of pages which will describe similar items like FAQ entries, meeting minutes, book reviews, ...

We believe that wiki templates are good handles for enriching the semantics of wiki pages, as it permit one to do so *without changing the editing mechanisms and workflow* already known by authors. Assuming a (wiki-compatible) way to enforce template matching can be found, we can add microformat capabilities to whatever wiki markup is used and then use templates to drive the creation of pages as authors have been done since the early wiki days. This compatibility with the pre-existing workflow of authors is a big win, and

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

[2]`http://www.wikipedia.org`

[2]In the wiki lingo the term "template" is frequently encountered, but it is used with various meanings. In [9] one of the most widespread interpretation is used, the *presentational template interpretation*: a template is a HTML page skeleton with holes to be filled by the wiki engine with the rendering of parts (e.g. title and body) of a wiki page content. In this paper we are concerned instead with the *content template interpretation*: a template is a mechanism to reuse, to some extent, wiki page content across different pages. This justifies our subsequent use of the term "template" to refer to content templating, unless otherwise stated.

can make the difference with previous attempts at semantically enriching the web, at least for the slice of it which is delivered through wikis.

In this paper we therefore study state of the art wiki templating mechanisms in order to understand if and how they can be used for the task of fostering the production of semantically rich web pages. In doing so we identify two alternative templating models which can be found in state of the art wiki engines: functional templating (in which templates are invoked by name and passed parameters) and creational templating (in which templates are simply copied as new content at the beginning of a page revision history). We will see that none of the two is up to the task and, more generally, that both models have drawbacks even in implementing basic content templating functionalities in a way that is compatible with foundational wiki principles. We then propose a new templating model, called *lightly constrained templating*, which fixes the drawbacks of the existing models and which we believe to be a suitable mechanism to ease the production of template-based (lowercase) semantic web pages.

It is worth emphasizing that our approach is sensibly different than traditional semantic wikis. We are not developing yet another semantic wiki, nor providing an interface to ease the input of explicit semantic metadata. We are rather showing how to extend traditional wikis with semantic capabilities, without sacrificing the well-established editing workflow.

Paper structure. To properly classify state of the art wiki templating models we start in Section 2 from a task analysis considering the wiki user goals that can be mapped to template-related tasks. Then, in Section 3, we detail functional and creational templating models, characterizing them on the basis of how the conceptual action "apply a template to a page" is made available to the user. Section 4 presents lightly constrained templating while Section 5 discusses our implementation and show its deployment possibilities in standard and semantic wikis. Section 6 concludes the paper and presents some related works.

2. TASK ANALYSIS

The abstract concept of wiki templating concept is frequently blurred to a specific templating mechanism implementation available in a wiki engine. For example, a Wikipedia user probably thinks that wiki templating as a whole is achieved through the creation of pages like the software infobox[3] template and their invocations from the markup of other pages. Similarly a MoinMoin[4] user hearing "wiki template" probably depicts in her mind the list of *seeding pages* she can choose from when clicking on a page creation dangling link.

To analyze templating in an engine-agnostic way, we performed a task analysis [13] of user activities that can benefit from templating support implemented by wiki engines. The results of our task analysis are summarized in Table 1. In the table we stick to some well-established terms [13]: a *goal* is a state of the system a user wishes to achieve, an internal task (or simply a *task*) is a sequence of one or more activities the user thinks are required to achieve a goal.

[3]`http://en.wikipedia.org/wiki/Template:Infobox_software`
[4]`http://moinmo.in/`

Table 1: Goals and tasks (with roles of users pursuing them) that can be mapped to actions on devices implemented by wiki templating mechanisms.

No.	*Role*	*Task / Goal*
1.	editor	instantiate a boilerplate page *goal*: (quickly) create a new page that is instance of another one
2.	tailor	create/modify a boilerplate page *goal*: publish a generic page to foster population of the wiki space with boilerplate instances, by easing their creation
3.	editor	copy and *modify* the content of an existing page (including adding/removing content) *goal*: (quickly) create a new page that is similar to another one
4.	editor	apply/unapply to a page, the content structure of other pages *goal*: uniform (distinguish) the content organization of a page with (from) other pages
5.	tailor	create a predefined content organization for future use *goal*: publish a reusable page component to foster population of the wiki space with wiki pages that share organization of content (parts)
6.	tailor	change at once the structure of several pages *goal*: (quickly) change the content organization of several pages to smooth visitors' experience when reading them
7.	editor	copy and *reorganize* the content of an existing page *goal*: create an alternative view of some content already present in the wiki

We will not bother the reader with the hierarchical task analysis of each task, since the more the tasks are decomposed, the tighter their dependencies become on a specific templating mechanism implementation. Yet we observe that all tasks can be eventually decomposed to actions on devices that are specific of some templating mechanisms, such actions will be presented in Section 3.

Each task of Table 1 is specific of a peculiar wiki user role (that notwithstanding the fact that the same user can play different roles at different times). The roles we have considered are the following:

visitor the most common user role, i.e. a user browsing the wiki simply to view pages authored by others. All template-related actions are hidden to such an user, so she is the actor of none of the tasks shown in Table 1. Yet visitors are relevant to our discussion, as users in other roles act *for* them, in particular to improve their user experience by uniforming the organization of pages;

editor a user that edits the source text of wiki pages either to create new pages from scratch or to change preexisting pages;

tailor users that are responsible of some wiki parts (i.e. page sets), for example of all the pages pertaining to a given subject, category, or which share other characteristics such as being co-located in the page namespace. Note that tailors are not the only authors of these parts of the wiki, but they are in charge of the customization of the contained pages [10], since otherwise the intrinsic editing freedom of the wiki would have been defeated.

Sometimes tailors have access to privileged technical features that are not available to other users. In such settings, according to folklore, users gain tailor (or "admin", as it is called in some wiki communities) privileges on the basis of reputation. Well-known examples of tailor expressions in real wiki communities are Wikipedia's bureaucrats[5] and administrators[6].

Task highlights. Task 1 and 2 (as we will see they often come in pairs) are both related to the common need of providing representations, in the form of several wiki pages, of objects of an universe that conceptually belong to the same class. For instance, a set of courses offered by computer science departments might need to be described by similar wiki pages, the same can be required for publishing periodically on the wiki reports of some recurring event (software releases, work meetings, football matches, . . .), or to "poll" users on a given subject.

Tasks 1 and 2 map to complementary activities pursued by a tailor and an editor. The former wants to foster the creation of several pages describing objects of the same class. Her strategy for achieving so, according to task 2, is preparing a generic page (*boilerplate page*) which represents a generic instance of the class of interest. The generality is expressed as page incompleteness: to obtain from the boilerplate an actual page (i.e. a page describing a class instance), the set of "holes" occurring in the page should be filled with instance-specific data. This strategy is the wiki workflow equivalent of the PROTOTYPE object-oriented design pattern [5]: an instance of a desired class is obtained by first copying an available prototype of the same class, and then filling in the missing information.

The issue of how to avoid that future edits on the instance page do not change "fixed" parts of the generic page, therefore breaking the class-instance relationship, is not trivial to achieve in some templating models (see Section 3). Task 1 is the point of view of an editor on the strategy of the tailor of task 2: to create an instance page of some class she chooses the corresponding prototypical page at page creation time (as it happens with seeding pages) and fills its holes.

Task 3 is similar, but acts on page pairs which are not related by a prototype-instance relationship: the editor wants to create a page which is only similar to another and, as it frequent happens, she starts doing so by copying and pasting the markup of the similar page. Obviously, in order to fulfil this task we require no templating mechanism device at all.

A different goal is that of enforcing a common content organization on several pages, in the hope of inducing a common look and feel and smoothing user experiences while browsing pages that share the same organization. Several examples of this need can be found in Wikipedia, where information/navigation boxes are made available by tailors to be used to describe conceptually similar parts of related pages. To name just a few of them: the software information box we mentioned earlier in this section, the box used to summarize the information of a discography album[7] and to navigate chronologically among several albums, the taxonomy box[8] associated to each animal page, . . . together with countless other.

Tasks 5 and 6 correspond respectively to tailor activities needed to setup a piece of reusable content structure, and to perform a batch change to all pages using it. Task 4 is the editor activity required to deal with a reusable content structure: use it in a page (by adding its invocation in the markup, possibly passing it parameters) or get rid of its usage (by removing the corresponding markup, possibly inlining template parts by hand).

Task 7 is related to a broader meaning of the word "templating", which considers the informative content—or the semantics, in the context of semantic wikis—of a page as a template to be applied to (the informative content of) other pages. Exotic as it may appears, scenarios in which this kind of templating is are infrequent. For example, in Wikipedia several conceptual sets of items do exist and are represented as several lists sorted with varying criteria. A paradigmatic case: from the Lists_of_countries[9] (note the plural "lists") page, several `list of countries by ...` pages are linked and are supposed to contain all the same countries, yet sorted differently. A way to see such an organization is to consider the List_of_countries[10] (singular "list") page as an informative content template page, and the other lists as different presentation of the same content.

3. TEMPLATING MODELS

The fact that wikis have increasingly been rising to complex content management systems (CMSs), in different domains and for different uses, should have led implementors to concentrate hardly on templating functionalities. Motivations and user needs behind templating have been discussed in the previous section, but a question is still unanswered: "how is wiki templating actually deployed?". We address it in this section.

A side objective of our research was to feel the pulse of the wiki clones with regard to that issue. Some very interesting surprises came out. First of all, we have discovered that a relatively small number of clones actually support content templating: in most cases clones use the term "templating" in the presentational template interpretation (see Section 1) to characterize customizable and interchangeable layouts and skins, more than the automatic generation of new pages based on existing ones. The good news is that content templating is supported by the most widespread wikis, with interesting differences.

Although different in the implementation details, all the templating models we are aware of can be framed into a single architecture, depicted in Figure 1. In that and subsequents figures, straight arrows represent wiki actions, with the distinction of *external actions* (initiated by a user and

[5]`http://en.wikipedia.org/wiki/Wikipedia:BUR`
[6]`http://en.wikipedia.org/wiki/Wikipedia:SYSOP`
[7]`http://en.wikipedia.org/wiki/Template:Infobox_album`
[8]`http://en.wikipedia.org/wiki/Template:Taxobox`
[9]`http://en.wikipedia.org/wiki/Lists_of_countries`
[10]`http://en.wikipedia.org/wiki/List_of_countries`

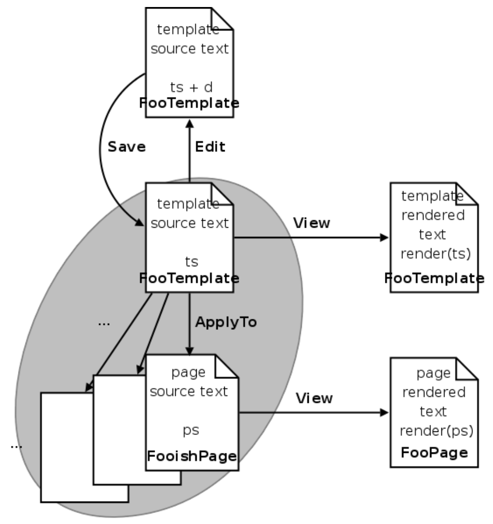


Figure 1: Template relationships with (common and template-specific) wiki actions in a generic templating model. The actual implementation of the ApplyTo action characterizes different templating models.

depicted with bold labels) and *internal actions*; dashed arrows represent generic relationship among wiki entities; the big black arrow represents the mapping of conceptual actions to their actual implementations.

Two kinds of pages are involved in template processing:

template (e.g. `FooTemplate` in Figure 1): the master page on which other pages can be based;

instance (e.g. `FooishPage`): the result page obtained by applying a master page to a target one.

The core of templating is the ApplyTo action. It consists of actually generating an instance page from a template (i.e. *applying* a template *to* a target page). Different implementations of ApplyTo characterize different templating models.

Before discussing these models, it is worth remarking that templates are normal wiki pages, which can be accessed and edited as any other page of the wiki site. Unlike "real" CMSs, where templates are treated as special resources, wikis consider templates and instances at the same level, and allow users to manage both kinds in (almost) the same way.

As a consequence, the View action can be applied to both template and instance pages: it produces a rendered page, where content is blended with the overall layout of a wiki site, and enriched with user interface elements and hyperlinks. Similarly, a Save action can be performed to create new versions of templates and instances. The Edit action presents some differences: it is a common wiki Edit for the templates, but can be limited to some extent when applied to the instances, as we will discuss in a while.

Narrowing the analysis to the ApplyTo action, we have identified two different templating models: *functional templating*, in which templates are invoked by name and possibly passed parameters, and *creational templating*, in which templates are simply copied as new pages at the beginning of instance revision histories.

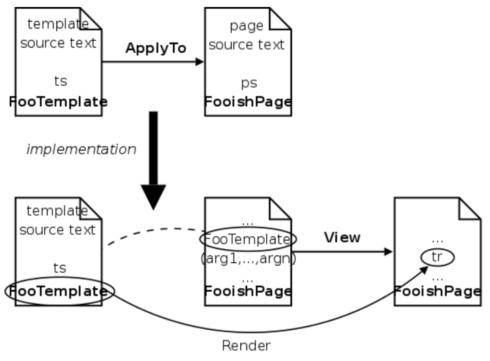


Figure 2: Functional templating: ApplyTo action implementation.

3.1 Functional Templating

The most common example of functional templating is implemented by MediaWiki[11] and heavily used in Wikipedia. A functional template is a page including a set of (possibly) named placeholders (or "holes") which will be substituted for actual values passed as formal parameters, when invoking the template for a specific content. For instance, most Wikipedia information boxes providing structured information about countries, sports, animals, and plants are created with such technique.

A functional template is applied invoking it by name and passing actual parameters. Both invocation and parameter passing are achieved with special purpose syntax in the wiki markup. Figure 2 summarizes this templating model.

The first characteristic of such templating is a strong and permanent connection between a template and its instances: whenever an instance page is displayed, the template is expanded on-the-fly and the result is in turn passed to the rendering engine of the wiki clone. There is no way to have individual instances differ from the original template other than in filling its named holes.

A second relevant aspect is about the source code of templates and instances. They are quite different: a template contains instructions and structures to organize the final content, while an instance contains the invocation of template by name and parameter passing.

As a consequence, the user experiences while editing templates or instances are sensibly different. It involves dealing with placeholders and (HTML or wiki-syntax based) structures in templates, and with parameters modifications in instances. In the latter, users can simply add, remove, or change parameters (which can be arbitrarily complex though) and cannot modify the overall structure of the template content, unless they are willing to directly modify the source of the template. That however is a different action to be performed on a different page, possibly requiring different access permissions as it would affect several other pages.

Another related editing problem in instances is the *non-linearity of the markup* with respect to its rendered form. Wiki editors can usually edit source text in wiki engines and syntaxes they do not know by spatial analogy: the desired editing point can be found comparing the rendered text of a page with its companion source text, possibly comple-

[11] `http://www.mediawiki.org/wiki/MediaWiki`

menting this with some plain text search. This cognitive process is possible only if the markup is *linear* with respect to the rendered text, i.e. only if a particular change (addition, modification, removal) to the source text maps to a similar change in its rendered form. Functional templates violate this property: if the user try to find its editing point looking at some rendered text that comes from the template, she has no hope to find the corresponding source text in the markup of the instance page.

With respect to the tasks discussed in Section 2 functional templating scores pretty well, as on top of its devices tasks 2, 4, 5, and 6 can be implemented straightforwardly. However, task 1 requires additional syntactic and conceptual knowledge, and produces as a result pages potentially troublesome for the average editor, due to markup non-linearity.

This brief analysis helps us in highlighting the most relevant pro/con aspects of functional templating. First of all, evident benefits derive from the strong connection between templates and instances: template matching is automatically enforced since pages are actually generated on-the-fly. Deviation from the original template are prevented and pages (or better, template fragments) are automatically and permanently uniform. Moreover changes on a template are automatically spread all over the instances so that managing set of pages with the same template is simple, fast, and reliable.

The creation and editing of page instances is another important advantage: authors do not need to master complex structures or deal with the overall organization of page content, but they simply have to provide a value for each parameter. Functional templating is then very useful for input of structured data, like table records.

On the other hand, the fact that the source content of an instance does not correspond to the actual template content arises some issues, in terms of adherence to the wiki philosophy. One of the most appreciated and comfortable features of a wiki is the full control over the content that is provided to authors, despite the simplicity of the syntax. Wiki users are not required to master complex languages and protocols, yet have fine-grained editing control over their pages. To some extent, such control is lost in the functional model since users pass their parameters to a black box which is in charge of re-organizing their presentation.

Users might be also interested in slight modifications to their pages in places other than the named holes, even if they are generated from a template. Those changes cannot be applied by strictly adopting a functional approach, since the modification on a template would be propagated over all the derived pages.

3.2 Creational Templating

The term "creational template" introduced in this paper indicates those pages (*seeding pages*) used as the starting point for the creation of new ones, with the same structure and initial content. Though seeding pages were available in the early wiki days, MoinMoin first re-introduced such templating model giving it wide-acceptance. Creational templating is now supported by other wiki clones: basically it consists of simply copying the whole source code of a page into a new one, and then editing the new page as usual. Creational templates are widely used to create (initially) uniform pages in a wiki site or to quickly generate new content from preexisting content. Figure 3 summarizes this model.

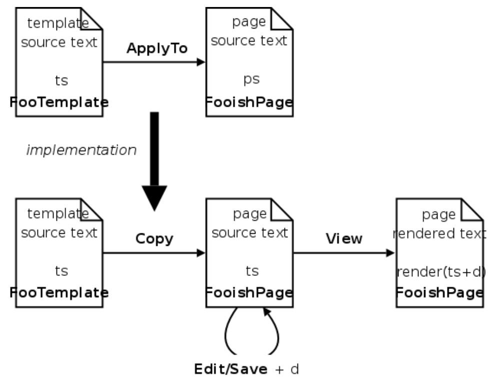


Figure 3: Creational templating: ApplyTo action implementation.

In several respects, creational templates are dual to functional templates. First of all, with creational templating, templates and instances are only *weakly* connected: once a page has been derived from a template, it lives as an independent entity within the wiki. It can be modified and twisted, up to become something completely different from the original source. Accordingly, two pages derived from the same template can end up having nothing in common. There is no way (although there would be many reasons) to connect a page and a template for the whole content life-cycle. Even though intuitively there is nothing preventing creational templates to keep back references to the template used to create a page, practically no wiki engine we are aware of exploits such back-pointers to avoid "excessive" drifts of instances from templates.

The second point is the fact that the source code of a template and its instances use the same syntax, and that such syntax can be mostly templating-agnostic. Modifying an instance means modifying directly the (HTML or wiki-syntax based) structures of the page. The content of an edited instance page is, in fact, directly passed to the rendering engine which transforms it into the final wiki page; no access to the template page is needed at VIEW-time. Thus, the EDIT action is template-agnostic both for instances and templates.

Editing a template, however, does not imply changes in the instances derived in the past from that template, as it happens in the functional model. Of course such changes do affect instances that will be created in the future.

Regarding the task analysis of Section 2, we observe that tasks 1, 2, 3 are straightforwardly implemented on top of creational templating devices. Task 5 is possible in theory, but it has to be built on top of an input mechanism that permits to choose a creational template while editing *part* of a page. To our best knowledge none of the wiki clones supporting creational templating implements this.[12]

[12]However, a workflow supporting such *local creational templating* is easily conceivable. Consider for example a wiki engine with support for section (i.e. heading-delimited page part) editing. When the user clicks on the edit link for an empty section, a creational template selection box can be presented to her. Then, the selected template can be used to fill only the section being edited instead of the whole page as it happens in other creational template capable engines.

As expected, the orthogonality with the functional approach is mirrored by pros and cons of creational templating. Creational templating does not provide any mechanism to enforce template-matching on future revisions of a page and so it makes possible (and very common) that radical changes happen between the initial instance and subsequent versions. Since the connection between a template and its instances practically disappears after the first Copy action, these templates do not guarantee uniformity among pages. For the same reason, users cannot modify with a single action all the pages derived from a template (i.e. task 6 is not mappable to actions on creational templating devices), and the management of set of pages still requires manual and error-prone interventions.

On the other hand, the fact that the output of the ApplyTo action is the final content (not yet rendered into the layout and formatting of the wiki site, but already organized in tables, lists, ...) is a point of strength for the creational approach. Users, in fact, have fine-grained control over the pages and they can easily customize template instances, saving the straightforwardness and speed of the wikis. Markup linearity with respect to the rendered text is not hindered by creational templating.

3.3 Templates in State of the Art Clones

Thus far our analysis of basic wiki templating capabilities has not ended up with the proclamation of a winning model. On the contrary, both functional and creational approaches proved to have benefits and drawbacks, according to user needs. It is no accident that they are differently supported by wiki clones.

In order to evaluate the actual support for wiki content templating, we have studied the most popular wiki engines. As popularity sources we have used the topmost entries (with a coverage of about 40% of the total) of authoritative wiki comparison web pages: c2.com wiki[13], wikimatrix[14], Wikipedia's page "Comparison of wiki software"[15], "wiki popularity"[16] page in WikiCreole. The URL references of the considered wiki engines as well as other details are available in a separate technical report [3]. Such an analysis gives us a clear picture of templating implementation in state of the art wiki clones.

A first fact is the different support between creational and functional templates: about a half of the clones adopt a creational approach, while less than a fifth adopt a functional one. Actually, full-fledged functional templating is supported only by MediaWiki (and Wikia which is MediaWiki-based). We have considered in the same class also other wikis, such as WackoWiki, TikiWiki, and OddMuse, which support functional templating without parameter passing (i.e. as some form of page inclusion). This choice is justified by the observation that pros/cons of functional templating are only marginally affected by parameter passing.

Similarly, XWiki allows designers to define patterns of structured information (stored as a different page, and called by an include-function) which can be re-used in any page.

Table 2: Templating model support in popular state of the art wiki engines.

Wiki Engine	*Creational*	*Functional*
AtlassianConfluence	✓	✗
DekiWiki	✓	✓
DidiWiki	✗	✗
DokuWiki	✓	✗
EclipseWiki	✗	✗
EditMe	✗	✗
ErfurtWiki	✗	✗
FlexWiki	✓	✓
InstikiWiki	✗	✗
JSPWiki	✗	✗
KwikiKwiki	✗	✗
MediaWiki	✓	✓
MicKI	✗	✗
MoinMoin	✓	✗
OddMuseWiki	✗	✓
OpenWiki	✓	✗
PerSpective	✓	✗
PhpWiki	✗	✗
PmWiki	✗	✗
ProntoWiki	✗	✗
SocialText	✓	✗
TeleparkWiki	✗	✗
TiddlyWiki	✓	✗
TikiWiki	✓	✓
TigerWiki	✗	✗
TWiki	✓	✗
UseModWiki	✗	✗
VeryQuickWiki	✓	✗
VimKi	✗	✗
WackoWiki	✗	✓
WakkaWiki	✓	✗
WiClear	✓	✗
Wikia	✓	✓
WikiWig	✗	✗
XwikiWiki	✗	✗
YaWiki	✓	✗
ZwiKi	✗	✗
Total: 37 (100%)	17 (≈ 46%)	7 (≈ 19%)

Pages containing such information can be edited and re-edited as simple HTML forms, and will be rendered as the original template. Even if the word "template" is not explicitly used by XWiki, and users do not access the source/text code of the invocation, such feature can undoubtedly be classified as a functional templating.

Also the class of creational templates includes some wikis which do not provide such templates in their original form (as MoinMoin does), but implement some differences. For instance DokuWiki provides different namespaces to cluster set of pages sharing a common skeleton. When a page is created, DokuWiki looks up whether there is a template for that namespace and copies it into the edit field for the new page. However, such template file cannot be edited directly through a web interface, but an administrator has to access and change a text file saved on the server. TWiki

From there on, the templating model is identical to that presented in this section.

[13] http://c2.com/cgi/wiki?TopTenWikiEngines

[14] http://www.wikimatrix.org/

[15] http://en.wikipedia.org/wiki/Comparison_of_wiki_software

[16] http://www.wikicreole.org/wiki/WikiPopularity

implements a creational approach natively for some pages (for instance, user profiles) and allows users to simulate it for other pages customizing the installation.

Creational templates are very often supported by wikis which integrate a WYSIWYG editor: for instance Confluence, TikiWiki, and PerSpective allow users to select a template, to create a new page based on that template, and to further change its content, without accessing the source code. Even in those cases, however, the template is only used for the first deployment of that page.

Finally, we also found many lists of requested features (for instance, for ZWiki or JSPWiki) where both creational (referred to as "a la MoinMoin") or functional ("a la MediaWiki") templates hold a relevant position. Our feeling is that there is still a strong willingness to support and improve wiki content templating.

4. BETTER CREATIONAL TEMPLATES

Our answer to the question of whether state of the art wiki templating model can be improved is affirmative. In this section we present a novel templating model, as an attempt to create a merger of the positive aspects of the two templating model presented in the previous section. Our model is not radically new, but rather a conceptual "patch" for creational templating.

Our starting point is maintaining access to the markup of the page as a fundamental feature of wikis. This has led us to base our proposal on top of creational rather than functional templating. The main drawback of creational templates is the lack of "retroactivity", induced by the weak connection between templates and instances: no change which breaks the original template can be prevented or notified. Our solution to this issue consists of integrating a non-invasive mechanism to validate *a posteriori* page content, so as to check whether the prototype-instance relationship has been violated. Our proposal is still a model of the architecture of Figure 1 (being a patch of the creational model) and exploits an instantiation of the light constraint framework [4], which is briefly described in the next section.

4.1 Light Constraints in a Nutshell

Light constraints [4] on wiki pages are, roughly speaking, desiderata on page contents. They are just "desiderata" since even when violated they do not inhibit users to access the page either for reading or writing, as opposed to hard constraints which do that. Light constraints tend to appear spontaneously as best practices in wiki communities, but also in domain-specific wikis where they are inherited from the given domain (e.g. mathematical correctness of proofs presented as wiki pages). Light constraints can also be thought as constraints whose enforcement is compatible with "The Wiki Way" workflow of viewing and editing pages.

Previous work [4] has detailed a framework for dealing with *light constraints* on the content of wiki pages, observing how *de facto* constraints spontaneously appear in wiki communities to encode best authoring practices or to account for the peculiar requirements of domain-specific wiki engines.

A framework for dealing with light constraints has been presented in the past [4]. In such a framework constraints are expressed as validator functions[17] that can be associated

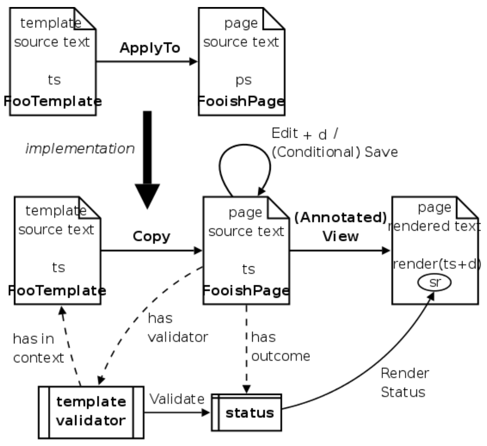


Figure 4: Lightly constrained templating: ApplyTo implementation and overall architecture.

to pages (in a many-to-many relationship) and used to check whether the constraints have been respected, a step possibly involving additional pages (the *validation context*). Failed validation attempts return sets of localized error messages.

Since saving cannot be inhibited when constraints are violated without diminishing authors' freedom, the framework uses *conditional saving* to present to the author validation errors (if any) as a single additional step before a SAVE action can be completed. The editor can then either ignore the errors and save the page anyway or perform additional editing round-trips to fix them. Similarly, the VIEW action is changed to *annotated viewing*: visitors viewing a page are shown the usual page content as well as validation information such as validators, contexts, and error messages. This way the internal state of the engine is fully disclosed to visitors and the efforts of WikiGnomes[18] can be driven (by tailors) to fix validation errors.

4.2 Lightly Constrained Templating

We use the term *Lightly Constrained Templates* (LC Templates) to indicate creational templates, empowered by light constraints on template matching. The key observation consists of encoding as a light constraint the fact that a page should match the template it has been generated from, and hence a given information pattern. Whenever a page is displayed or saved, the system verifies if that template is matched and notifies the validation status to the users. Figure 4 shows this approach.

Each page is associated to a set of validators which take as input the page content a validation context. A validation context is a set of pages required for validation: in this case, it is only the original template which generated the current page. The validator produces either a "ok" message if the template matches the instance, or a list of localized errors (indicating where mismatches occur) when it does not. According to our general architecture the VIEW action be-

[17]this means that light constraints are described procedurally and externally to the wiki page space; they are associated declaratively with pages (by tailors). The alternative approach to use domain specific languages to describe declaratively light constraints, together with pros/cons of the two approaches, is briefly discussed in [4]

[18]`http://en.wikipedia.org/wiki/Wikignomes`

comes an annotated viewing (the page contains both the actual content and the validation report), while the SAVE action becomes a conditional save (the report is displayed before saving, and users can choose to ignore or take it into account, by fixing mismatches).

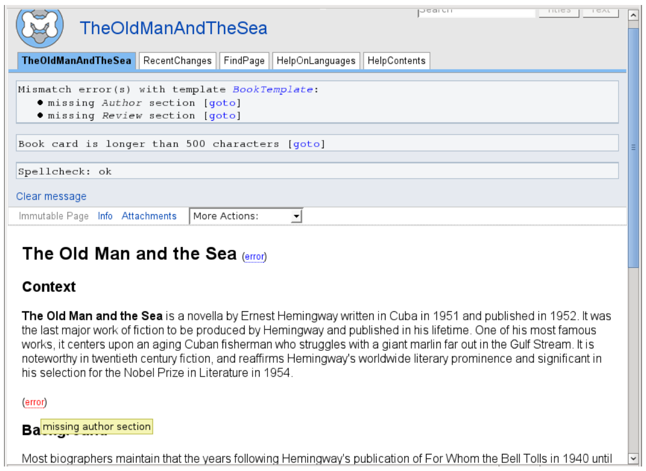
It is worth observing that such architecture is independent from the granularity and power of the validation process. Different solutions can be implemented: from regular expression matching to search over predefined areas, up to advanced templating on the informative content of a page. In Section 5 we will discuss a possible spectrum of validators. What is relevant here is the fact that such validators improve the templating model without interfering with the open wiki philosophy.

Our model implements a permanent connection between templates and their instances, this is achieved through the introduction of light constraints. Although a page is not actually generated from a template on every VIEW (as happens in the functional approach), the validation process guarantees a permanent relation between those pages. As an additional benefit LC templates enable *template re-targeting*. Given that the instance-template relationship is kept in the validation context, and given that the validation context can be changed by tailors, it is possible to change the template applied to a page even after the page's creation time. If the new template still matches the instance page no constraint violation will be reported, otherwise all mismatches will be shown to the tailor and, for those left unfixed, to editors and visitors.

Note also that LC templating allows tailors to modify a template and consequently modify all the derived pages. The general light constraint architecture provides for a daemon in charge of notifying changes of a page to all the pages which include it in their validation context. Applied to this domain, it means that a modification on a template is spread all over the wiki. Whenever a page "connected" to that template will be displayed, a notification message will be included in that page. Then users can spontaneously work to fix the mismatch. There is a subtle but important difference with functional templating here: while function templating *enforces* template-matching (at the price of limiting editing freedom) our approach *verifies and notifies* mismatches. Users can ignore notifications, and go on saving their content. At the same time, they have complete freedom and full information on their content. In a sense we are decentralizing batch changes exploiting the primary power of wiki communities: collaboration.

The linearity of markup is another positive aspect of LC templating. Being substantially a patched version of the creational templating, our approach provides a correspondence between the source of a page and its rendered version. Users access the same content, but also receive information about the templates and, if needed, can apply changes and customization. Another interesting property derives from the decoupling of validation and rendering. Since validation is performed outside the wiki (even if a wiki module performs that operation, it remains logically independent), different validators can run (as we discuss later on) without changing the overall architecture of the wiki. Moreover the same validator can be potentially re-used in different wiki clones.

Summarizing, we believe LC templating to be the best trade-off for wiki content templating in a generic setting. Specific settings, such as strong needs of structured editing, might still consider the adoption of functional templating (see Section 6). The need of templating page parts, as opposed to whole pages, can be easily addressed with LC templates by implementing the local creational templating workflow proposed in Section 3.2.



```
Mismatch error(s) with template BookTemplate:
    • missing Author section [goto]
    • missing Review section [goto]

Book card is longer than 500 characters [goto]

Spellcheck: ok
```

The Old Man and the Sea (error)

Context

The Old Man and the Sea is a novella by Ernest Hemingway written in Cuba in 1951 and published in 1952. It was the last major work of fiction to be produced by Hemingway and published in his lifetime. One of his most famous works, it centers upon an aging Cuban fisherman who struggles with a giant marlin far out in the Gulf Stream. It is noteworthy in twentieth century fiction, and reaffirms Hemingway's worldwide literary prominence and significant in his selection for the Nobel Prize in Literature in 1954.

(error)

Ba missing author section

Most biographers maintain that the years following Hemingway's publication of For Whom the Bell Tolls in 1940 until

Figure 5: Screenshot of our LC templating implementation in the MoinMoin wiki engine. The shown tooltip is associated to the first of the template mismatch errors summarized at the top of the page, localized at the second `(error)` marker in the page.

5. LC-TEMPLATES IMPLEMENTATION

We have implemented LC templating on top of the MoinMoin wiki engine, a screenshot of our implementation in action can be seen in Figure 5. At the time of writing we have not yet implemented any specific microformat syntax, but we believe this to be the easiest part of the problem at hand. We have therefore rather chosen to fully implemented the light constraint architecture of [4], and the template matching mechanism as a specific validator.

In Figure 5 it can be seen that the page is about a review of Hemingway's book "The Old Man and the Sea". The page itself is an instance of the template `BookTemplate` and this is reflected in the page having a specific template matching validator and the (template) page `BookTemplate` in its validation context. Validator and context are automatically associated to the page when it is firstly created using the usual Moin workflow for this, and are then stored as part of the page markup using Moin's `#pragmas`. This choice enables template re-targeting by simply changing the template name in the page markup and is also the preferred Moin's place where to keep page meta-information.

The box at the top of Figure 5 shows the validation outcome for the validators associated to the page. The latter two validators show that template matching is smoothly integrated with the generic validation infrastructure of [4]. The former validator is the template matcher validator and is spotting two localized errors, namely that sections "Author" and "Review" are missing in the page markup. This is the peculiar aspect of our solution: authors are notified that parts of the original templates are missing and are encouraged to fill in the appropriate information.

We do not hope to have our implementation adopted by the majority of the wiki engine deployed on the web right away, therefore in the remainder of this session we discuss deployment techniques for LC templates in wikis for which an instance of the light constraint framework is available.

5.1 Syntactic Template Matching

With the light constraints machinery already in place, the task at hand is brought to implement a single validator whose specification is as follows.[19] The validator can be associated to instance pages and relies on a singleton validation context containing the template that has to match the instance. In fact it is recommended that in a LC deployment the association is added automatically by the engine when a page (or a section, in case of local templating) is generated creationally choosing from a list of the available templates. Of course tailors are free in the future to remove the association or to change it, re-targeting the instance page. The validator will be invoked on instance pages at (conditional) SAVE-time, or alternatively in a batch fashion (if a batch validator is available) or, as a last resort, at VIEW-time, depending on the details of the light constraint machinery implementation. Its output, the *validation outcome*, is either an assessment that the instance matches its template or a list of error messages containing the string `"template mismatch"` and localized where the mismatches happen.

Fulfilling the above specification implementing the corresponding function is straightforward. We only require that a special markup syntax, a marker, is available for denoting where holes occur in the source text of a template page. Such a template can be converted to a non deterministic regular expression by replacing hole markers with blocks of "any character sequence" (which would be `.*` in popular regular expression syntaxes). The obtained regular expression can be applied (with the appropriate flags such as multi-line support and with appropriate string delimiters) to instance pages to check for template matching.

If the regular expression matches an instance it is safe to return a validity assessment. If it does not, a naive assumption is that the page is not an instance of the template. In order to be less naive, a few additional steps in the generation of the regular expression can be be taken. For example the template page can be parsed by the legacy wiki engine page parser and "any character sequence" blocks (or more specific blocks like "any blank character") can be inserted where appropriate.

Localization of mismatches can be easily implemented as well. Indeed atomic `.*`-delimited blocks of the generated regular expressions can be wrapped in optional groups `(...)?` (a feature that is supported by all major regular expression implementations) and a posteriori checking for empty groups can be used to identify where (i.e. at which character position) an expected markup structure is missing. Yet easier, PCRE (the most widespread regular expression engine) supports *callouts*[20], external functions that can be attached to regular expression markers and are invoked when the engine encounter them. Callouts are passed the current position of the underlying automata on the input string and can store that information for further processing.

[19]We recall that a *validator* is usually a simple function written in the implementation language of the wiki engine and loaded as an external plugin.

[20]`http://www.pcre.org/pcre.txt`

To also achieve "named" error messages for template mismatches, in our implementation we added, using Moin's macros, optional labels to the fixed text parts of a template. If the match failure is due to the lack of such a fixed text part then an error can be reported stating that a given labeled section is missing (which is precisely what is happening in Figure 5).

5.2 Semantic Template Matching

Though our proposal is in apparent contrast with semantic wikis, it is worth observing that LC templating has synergies with them. In this section we discuss some of them.

In [4] it is discussed how validators can benefit from the availability of page semantics to perform validity checks which are more intimately related to the informative content of a page than to its presentational structure alone. This aspect, reconsidered in the LC templating setting, leads to a scenario in which the content structure of a template page is its semantic: what is the "meaning" of a content template if not the very same set of content structures which builds it up? Similar structural meanings have already been studied both in the document engineering community [2] and for wiki-specific needs [12]. Once such kind of meanings are available the template matching validator would be more reliable, being capable of abstracting over string matching and syntactic ambiguities.

An added benefit of LC templating coupled with semantic annotations would be the ability to implement devices that support task 7 of Table 1, which has been left unsupported by the templating models discussed so far. Indeed we can push forward the above-mentioned idea and consider the (non-structural) meaning of a page, i.e. the set of its semantic annotations, as a novel templating content. We call it *meaning templating* and we like to place it in the far extreme of the following templating spectrum:

$$\begin{array}{c}\text{presentational}\\\text{templating}\end{array} \longrightarrow \begin{array}{c}\text{content}\\\text{templating}\end{array} \longrightarrow \begin{array}{c}\text{meaning}\\\text{templating}\end{array}$$

where being more on the right corresponds to a higher degree of machine understanding of templates. With meaning templating the informative content of a page is encoded as semantic annotations and semantic verification among the informative contents of two pages can be performed. This can be exploited for example to ensure (to fulfill the Wikipedia needs of Section 2) that two pages contain the same set of items, notwithstanding their orders.

6. CONCLUSIONS AND RELATED WORK

In this paper we have argued that wiki content templating is a good handle to use for the diffusion of semantically rich web pages. We believe that a clever use of templating can succeed where other approaches failed, namely in convincing authors to enrich their wiki pages. We believe so because we are not requiring them to change their editing habits or mechanisms. However, to be successful we need two tools: wiki markup support for microformats and a suitable templating technology. The former should be trivial to achieve, will be subject of our future work, and we are quite sure that in the forthcoming years it will be adopted spontaneously by the main actors in the wiki clone playground. The latter requires some background study and it is exactly what we have delivered with this paper: a systematic study of state of the art wiki templating models which have spot some deficiencies, an improved model which has fixed them, and an easy to implement (as shown by our own realization) deployment technique for the proposed model.

To our best knowledge, the wiki literature is mostly devoid of works on wiki-specific templating needs, with the notable exception of Wiki-Templates [6]. The authors propose their own templating mechanism, specially crafted to support editing of structured wiki pages. Our analysis in this paper is more general and the models we have discussed are applicable both to structured and structureless wiki pages. An additional contribution of [6] is the identification of a set of requirements for supporting editing of structured wiki pages. It is interesting to discuss how our templating model scores with respect to such requirements. On the basis of the following considerations we claim that LC templating, in spite of not having being designed for structured editing, does fulfill most of its needs.

As expected, LC templating fails to properly address "R1: structure" (i.e. helping users to organize structured data), since it is structure-agnostic. However, if the pages are structured *per se*, adopting LC templating does not pose any additional problem: structure can be copied when templates are applied, and structure-specific user interfaces can be used for editing the templates themselves (e.g. the editing of the `edit template` of Wiki-Templates) and their instances (e.g. the complex web forms generated from `edit templates` of Wiki-Templates). Structure and LC templates can even exhibit synergism since structure, if expressed as the semantics of a given page, can be exploited to write validators with a deep understanding of page content.

With respect to requirement "R2: readability" (i.e. making easier to access content) we believe LC templates perform better than Wiki-Templates due to markup linearity. In Wiki-Templates a user willing to contribute to a page associated to an `edit template` (the template which has generated the input data form) is likely to experience interaction breakdown [15] either if it comes from a wiki engine other than Wiki-Templates or if the `edit template` has been changed. In a sense we are convinced that `edit templates` may hinder habit formation: they are probably worthwhile if structured editing is a major goal, but not in a more generic setting. LC templating on the contrary ensures markup linearity helping the user in finding the desired editing point in the source text by spatial analogy. Similar arguments holds for Wiki-Templates `display templates`.

Requirement "R3: safety regarding unintended changes" (i.e. making users aware of the effects of their edits) is fulfilled by LC templates which can enforce safety by the mean of validation (see Section 5). Finally, requirements "R4: tailoring" (i.e. allowing users to customize their content structures) and "R5: sharing of tailorings" (i.e. allowing users to share their templates) are satisfied for free in LC templating since templates live in the same space of ordinary pages. As such they can be published and customized by tailors without the need of implementing new conceptual wiki actions.

Future work. On the topic of LC templating and its synergy with (lowercase) semantic web, future work still needs to be done. As anticipated, an implementation of real-life microformats as specific wiki syntaxes, or rather piggy-backed on existing syntaxes (e.g. that of tables), is needed, together with some user testing. We also plan to investigate the differences (if any) between using microformats within wikis and within other, more constrained web authoring environments.

7. REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284:28–37, 2001.

[2] A. Di Iorio, D. Gubellini, and F. Vitali. Design patterns for descriptive document substructures. In *Proceedings of the Extreme Markup Conference*, 2005.

[3] A. Di Iorio, F. Vitali, and S. Zacchiroli. Templating wiki content for fun and profit. Technical Report UBLCS-2007-21, Department of Computer Science, University of Bologna, 2007.

[4] A. Di Iorio and S. Zacchiroli. Constrained wiki: an oxymoron? In *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*, pages 89–98, New York, NY, USA, 2006. ACM Press.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, 1995.

[6] A. Haake, S. Lukosch, and T. Schümmer. Wiki-templates: adding structure support to wikis on demand. In *WikiSym '05: Proceedings of the 2005 international symposium on Wikis*, pages 41–51, New York, NY, USA, 2005. ACM Press.

[7] R. Khare. Microformats: The next (small) thing on the semantic web? *IEEE Internet Computing*, 10(1):68–75, 2006.

[8] R. Khare and T. Çelik. Microformats: a pragmatic path to the semantic web. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, New York, NY, USA, 2006. ACM.

[9] B. Leuf and W. Cunningham. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley Professional, pap/cdr edition, Apr. 2001.

[10] A. MacLean, K. Carter, L. Lövstrand, and T. Moran. User-tailorable systems: pressing the issues with buttons. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, New York, NY, USA, 1990. ACM Press.

[11] E. Oren. Semperwiki: a semantic personal wiki. In *Proc. of 1st Workshop on The Semantic Desktop*, Galway, Ireland, 2005.

[12] E. Oren and M. Völkel. Towards a wiki interchange format (wif). In *Proceedings of the First Workshop on Semantic Wikis*, 2006.

[13] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey. *Human-Computer Interaction: Concepts And Design*. Addison Wesley, 1st edition, Apr. 1994.

[14] RDFa primer: embedding structured data in web pages. W3C Editor's Draft, 26 October 2007, `http://www.w3.org/2006/07/SWD/RDFa/primer/`.

[15] Y. Rogers. Coordinating computer-mediated work. *Computer Supported Cooperative Work (CSCW)*, 1(4), Dec. 1993.

[16] A. Souzis. Building a semantic wiki. *IEEE Intelligent Systems*, 20(5):87–91, 2005.

[17] M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer. Semantic wikipedia. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, New York, NY, USA, 2006. ACM.